1. **What is Micro Service?**

   Microservices are the small services that work together.
   Microservices are small and independent units and runs on its own process and database.
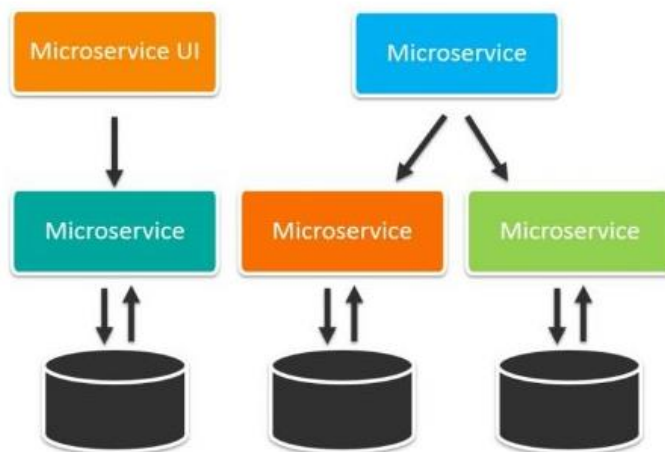
2. **Explain the Micro Services Architecture?**

   Microservice architecture, aka micro services, are a specific method of designing software systems to structure a single application as a collection of loosely coupled services.

   Each component of a microservice architecture has,

   - ✓ Its own CPU
   - ✓ Its own runtime environment
   - ✓ Often, a dedicated team working on it, ensuring each service is distinct from one another
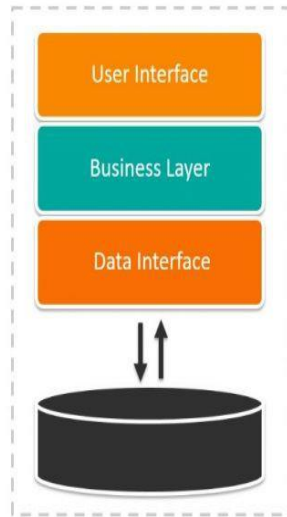

3. **Differentiate monolith and Micro Services architecture?**

   The monolithic architecture pattern has been the architectural style used in the past. In a monolithic architecture, the software is a single application distributed on a CD-ROM, released once a year with the newest updates. Examples are Photoshop. Monolithic structures make changes to the application extremely slow. Modifying just a small section of code can require a completely rebuilt and deployed version of software.
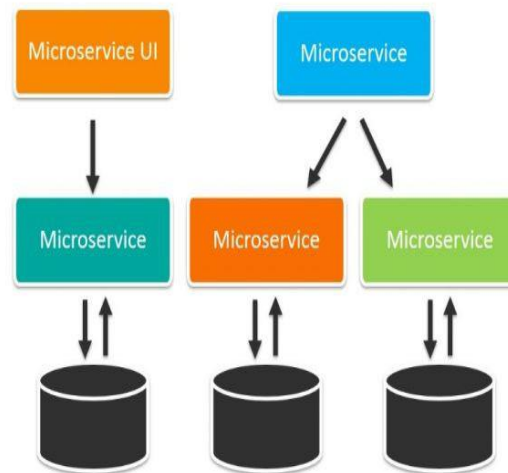
   Each component of a microservice architecture has,

- ✓ Its own CPU
- ✓ Its own runtime environment
- ✓ Often, a dedicated team working on it, ensuring each service is distinct from one another

4. Difference between Monolith application and Micro Service?

Monolith	Micro Services
Monolithic architecture is built as one large system and is usually one code-base	Microservices architecture is built as small independent module based on business functionality
It is not easy to scale based on demand	It is easy to scale based on demand.
It has shared database	Each project and module has their own database
Large code base makes IDE slow and build time gets increase.	Each project is independent and small in size. So overall build and development time gets decrease.
It extremely difficult to change technology or language or framework because everything is tightly coupled and depend on each other	Easy to change technology or framework because every module and project is independent
You must redeploy the entire application on each update.	Specific Micro Service to be deployed
If any of the module failed total application modules functionality can failed	If any of the module failed it won't impact others.
Development time is high	Development time is less

5. **Difference between SOA and Micro Service?**

SOA	Micro Services
Designed to share resources across services	Designed to host services which can function independently
Frequently involves component sharing	Typically does not involve component sharing
Involves sharing data storage between services	Each service can have an independent data storage
Better for large scale integrations	Better for smaller and web-based applications
Less flexibility in deployment	Quick and easy deployment
SOA is focused on application service reusability	Microservices are more focused on decoupling.

6. **Explain the principles of Micro Services?**

- ✓ **Single Responsibility:** The single responsibility principle states that a class or a module in a program should have only one responsibility. Any microservice cannot serve more than one responsibility, at a time.
- ✓ **Modeled around business domain:** Microservice never restrict itself from accepting appropriate technology stack or database. The stack or database is most suitable for solving the business purpose.
- ✓ **Isolated Failure:** The large application can remain mostly unaffected by the failure of a single module. It is possible that a service can fail at any time. So, it is important to detect failure quickly, if possible, automatically restore failure.
- ✓ **Infrastructure Automation:** The infrastructure automation is the process of scripting environments. With the help of scripting environment, we can apply the same configuration to a single node or thousands of nodes. It is also known as configuration management, scripted infrastructures, and system configuration management.
- ✓ **Deploy independently:** Microservices are platform agnostic. It means we can design and deploy them independently without affecting the other services.

7. **Explain advantages and disadvantages of Micro Services?**

Advantages of Microservices:

- ✓ Microservices are self-contained, independent deployment module.
- ✓ Microservices are independently manageable services. It can enable more and more services as the need arises. It minimizes the impact on existing service.
- ✓ It is possible to change or upgrade each service individually rather than upgrading in the entire application.
- ✓ Faster release cycle.
- ✓ Dynamic Scaling
- ✓ Less dependency and easy to test.

Disadvantages of Microservices

- ✓ There is a higher chance of failure during communication between different services.
- ✓ Difficult to manage a large number of services.
- ✓ The developer needs to solve the problem, such as network latency and load balancing.
- ✓ Complex testing over a distributed environment.

8. **Explain the Components of MicroServices?**
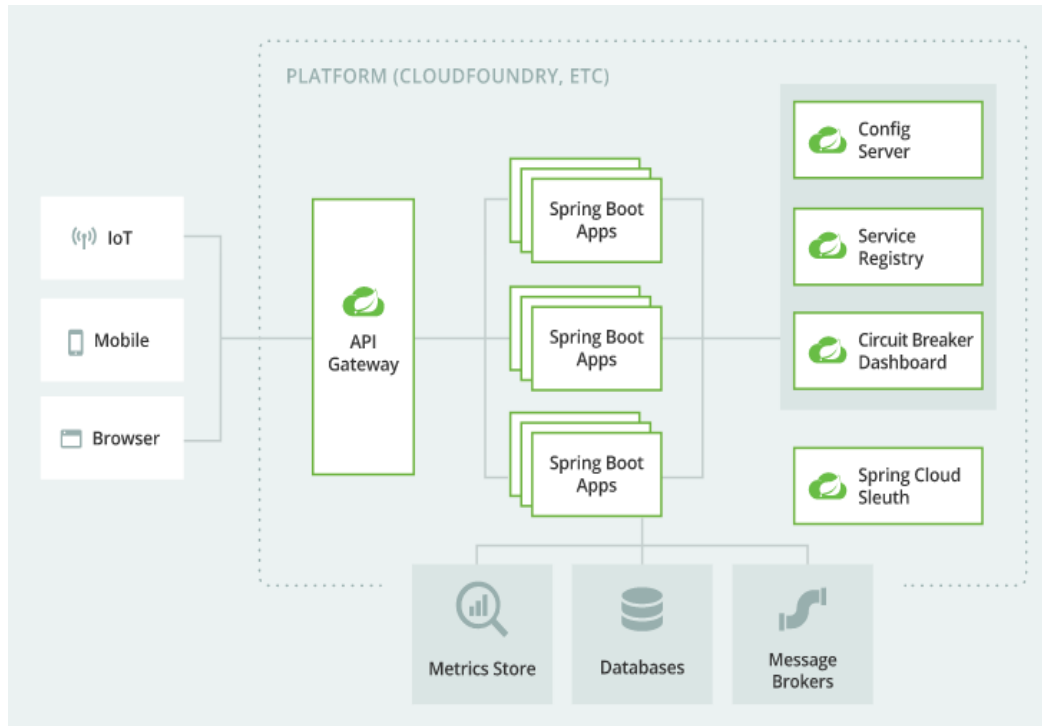
Spring Cloud Config Server: Responsible for maintaining the properties/yml files specific to all services. With the help of this we can remove the duplicate properties among services. Its resolve the problem of profile specific files also. We can enable the Spring Cloud Config Server by using the annotation **@EnableConfigServer**.

Netflix Eureka Naming Server: Eureka naming server is an application that holds information about all client service applications. Each microservice registers itself with the Eureka naming server. The naming server registers the client services with their port numbers and IP addresses.

This server will run on default port number 8761.

Eureka naming server fills the gap between the client and the middle tier load balancer.

For Example we already have two instances of CurrencyExchangeService1 and CurrencyExchangeService2 running on 8000, 8001

Suppose that we want to start another instance of currency-exchange-service that is CurrencyExchangeService3 and launch it on port 8002. Here a question arises, will ribbon be able to distribute the load to it?

The Eureka naming server comes into existence when we want to make maintenance easier. All the instances of all microservices will be register with the Eureka naming server. Whenever a new instance of a microservice comes up, it would register itself with the Eureka naming server. The registration of microservice with the naming server is called Service Registration.

Whenever a service wants to talk with another service, suppose CurrencyCalculationService wants to talk to the CurrencyExchangeService. The CurrencyCalculationService first talk with the Eureka naming server. The naming server provides the instances of CurrencyExchangeService that are currently running. The process of providing instances to other services is called Service Discovery.

Service registration and service discovery are the two important features of the naming server.

Hystrix Server: Hystrix server acts as a fault-tolerance robust system. It is used to avoid complete failure of an application. It does this by using the Circuit Breaker mechanism. If the application is running without any issue, the circuit remains closed. If there is an error encountered in the application, the Hystrix Server opens the circuit. The Hystrix server stops the further request to calling service. It provides a highly robust system.

Consider a scenario in which six microservices are communicating with each other. The microservice-5 becomes down at some point, and all the other microservices are directly or indirectly depend on it, so all other services also go down.

The solution to this problem is to use a fallback in case of failure of a microservice. This aspect of a microservice is called fault tolerance.

Fault tolerance can be achieved with the help of a circuit breaker. It is a pattern that wraps requests to external services and detects when they fail. If a failure is detected, the circuit breaker opens. All the subsequent requests immediately return an error instead of making requests to the unhealthy service. It monitors and detects the service which is down and misbehaves with other services. It rejects calls until it becomes healthy again.

Netflix Zuul API Gateway Server: Zuul Server is an API Gateway application. It handles all the requests and performs the dynamic routing of microservice applications. It works as a front door for all the requests. It is also known as Edge Server.

Zuul is built to enable dynamic routing, monitoring, resiliency, and security.

Zuul provides a range of different types of filters that allows us to quickly and nimbly apply functionality to our edge service.

Authentication and Security: It provides authentication requirements for each resource.
Insights and Monitoring: It tracks meaningful data and statistics that give us an accurate view of production.
Dynamic Routing: It dynamically routes the requests to different backed clusters as needed.
Stress Testing: It increases the traffic to a cluster in order to test performance.
Load Shedding: It allocates capacity for each type of request and drops a request that goes over the limit.
Static Response Handling: It builds some responses directly at the edge instead of forwarding them to an internal cluster.

Netflix Ribbon: It provides the client-side balancing algorithm. It uses a Round Robin Load Balancing.

Zipkin Distributed Server: Zipkin is an open-source project m project. That provides a mechanism for sending, receiving, and visualization traces.

Seluth: Responsible to maintain the unique id from request start to completion.

The Spring Cloud Sleuth token has the following components:

Application name: The name of the application that is defined in the properties file.
Trace Id: The Sleuth adds the Trace Id. It remains the same in all services for a given request.
Span Id: The Sleuth also adds the Span Id. It remains the same in a unit of work but different for different services for a given request.

9. **How to setup the Spring Cloud Config Server give an example?**

   ✓ Add below dependency in pom.xml,

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-config-server</artifactId>
</dependency>
```

   ✓ Add the below annotation on main class of the Spring Boot application

```
@SpringBootApplication
@EnableConfigServer
public class InternetBankingCloudConfigServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(InternetBankingCloudConfigServerApplication.class, args);
    }

}
```

✓ Add the below things in application.yml file get the properties/yml files from github

```
spring:
  application:
    name: internet-banking-cloud-config-server
  cloud:
    config:
      server:
        git:
          uri: https://github.com/full-stack-org/internet-banking-full-stack-react-micro-services/tree/main/internet-banking-git-config-files

server:
  port: 8888
```

✓ Default port for the Spring Cloud Config Server is 8888

10. Explain about Cloud Config Client and given example to get the details from Config Server?

Cloud Config Client is responsible for reading the details from Cloud Config Server

✓ Add below dependency in pom.xml file of micro services

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
```

✓ Add the below bootstrap.yml file to get the properties file or content from Config server

```
bootstrap.yml ⊠
1  spring:
2    cloud:
3      config:
4        uri:
5          - http://localhost:8888
```

11. Give an example for Netflix Eureka Server configuration?
✓ Add below dependency in pom.xml file of Eureka server micro service

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
</dependency>
```

✓ Add the below annotation on main class of the Spring Boot application

```java
@SpringBootApplication
@EnableEurekaServer
public class InternetBankingEurekaServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(InternetBankingEurekaServerApplication.class, args);
    }

}
```

✓ Add the below things in application.yml file to note it as Eureka Server

```yaml
spring:
  application:
    name: internet-banking-eureka-server

server:
  port: 8761

eureka:
  client:
    fetch-registry: false
    register-with-eureka: false
```

✓ Default port of Eureka Server is 8761

12. Give an example for Eureka Client?

✓ Add below dependency in pom.xml file of micro service

```xml
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

✓ Add the below annotation on main class of the Spring Boot application

```java
@SpringBootApplication
@EnableEurekaClient
public class InternetBankingExternalAccountsApplication {

    public static void main(String[] args) {
        SpringApplication.run(InternetBankingExternalAccountsApplication.class, args);
    }

}
```

✓ Add the below things in application.yml file to note it as Eureka client

```yaml
eureka:
  client:
    fetch-registry: true
    serviceUrl:
      defaultZone: http://localhost:8761/eureka
```

13. Given an example for Netflix ZuulApiGateway?

✓ Add below dependency in pom.xml file of ZuulApiGateway micro service

```xml
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
</dependency>
```

✓ Add the below annotation on main class of the Spring Boot application

```java
@SpringBootApplication
@EnableZuulProxy
public class InternetBankingZuulApiGateWayApplication {

    public static void main(String[] args) {
        SpringApplication.run(InternetBankingZuulApiGateWayApplication.class, args);
    }

}
```

✓ Add the below filter in micro service that should extend ZuulFilter

```java
@Slf4j
@Component
public class ZuulApiGateWayFilter extends ZuulFilter {

    @Override
    public boolean shouldFilter() {
        return true;
    }

    @Override
    public Object run() throws ZuulException {
        HttpServletRequest httpServletRequest = RequestContext.getCurrentContext().getRequest();
        log.info("Request is -> {} , Request uri -> {}", httpServletRequest, httpServletRequest.getRequestURI());

        return null;
    }

    @Override
    public String filterType() {
        return "pre";
    }

    @Override
    public int filterOrder() {
        return 1;
    }

}
```

✓ Add the below properties to maintain the micro services timeouts

```yaml
spring:
  application:
    name: internet-banking-zuul-api-gate-way

server:
  port: 8765

zuul:
  host:
    connect-timeout-millis: 15000000
    socket-timeout-millis: 15000000
    connection-request-timeout-millis: 15000000

eureka:
  client:
    eureka-connection-idle-timeout-seconds: 40
    eureka-server-connect-timeout-seconds: 40
    eureka-server-read-timeout-seconds: 40

ribbon:
  eureka:
    enabled: true
  ReadTimeout: 9000000
  ConnectTimeout: 9000000

hystrix:
  command:
    default:
      execution:
        isolation:
          thread:
            timeoutInMilliseconds: 15000000
```

✓ We should register the ZuulApiGateway as Eureka client to check the registered micro services under Eureka Server.
✓ Default port for ZuulApiGateway is 8765

14. Give an example for Hystrix?

✓ Add below dependency in pom.xml file of micro service

```xml
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>
</dependency>
```

✓ Add the below annotations main class of the Spring Boot application

```
@SpringBootApplication
@EnableHystrix
@EnableHystrixDashboard
public class AddingAllJarsApplication {

    public static void main(String[] args) {
        SpringApplication.run(AddingAllJarsApplication.class, args);
    }

}
```

✓ To handle the fallback of failure methods we need to add the below in controller code

```
@HystrixCommand(groupKey = "boot app", commandKey = "boot app", fallbackMethod = "fallBackFailed")
@GetMapping("/getData")
public String getName() {
    String responseFromRestServicesApp = restTemplate.getForObject("http://localhost:8888/api/v1/sayHi",
            String.class);
    String responseFromReactFullStackApp = restTemplate.getForObject("http://localhost:9999/api/v1/getData",
            String.class);
    return responseFromRestServicesApp + "\n" + responseFromReactFullStackApp;
}

public String fallBackFailed() {
    return "Fall Back Failed";
}
```

✓ In the fallback method if it is string we should return the response as string. If it is object we should return the response as object response. ==For example if we are expecting the response as Student bean having fields as id, name in the fallback methods we should set the id and name data in fallback method and we should return that response.==

✓ Hystrix Dashboard jar is used to track the error, success information of the applications. We need to get the details like http://localhost:9098/hystrix.stream

Hystrix Stream: http://localhost:9098/hystrix.stream

Circuit   Sort: Error then Volume | Alphabetical | Volume | Error | Mean | Median | 90 | 99 | 99.5

callStudentServiceAndGetData

10 | 0 | 0.0 %
0 | 0
0 | 0

Host: **1.0/s**
Cluster: **1.0/s**
Circuit Closed

Hosts	1	90th	115ms
Median	29ms	99th	225ms
Mean	55ms	99.5th	225ms

Thread Pools   Sort: Alphabetical | Volume |

StudentServiceDelegate

Host: **1.0/s**
Cluster: **1.0/s**

Active	0	Max Active	1
Queued	0	Executions	10
Pool Size	10	Queue Size	5

15. Give an example for micro services communication?

- ✓ Create two micro services

```yaml
spring:
  application:
    name: internet-banking-account-opening-service

server:
  port: 8084
```

```yaml
spring:
  application:
    name: internet-banking-forgot-password

server:
  port: 8082
```

- ✓ In the above example invoking the `internet-banking-account-opening-service` from `internet-banking-forgot-password`
- ✓ Add the below dependencies in pom.xml file

```xml
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

- ✓ Add the below annotation on main class of the Spring Boot application

```java
@SpringBootApplication
@EnableEurekaClient
@EnableFeignClients
public class InternetBankingForgotPasswordApplication {

    public static void main(String[] args) {
        SpringApplication.run(InternetBankingForgotPasswordApplication.class, args);
    }

}
```

- ✓ Add the below annotations in consumer class while invoking the provider service

```
@FeignClient(name = "internet-banking-zuul-api-gate-way")
@RibbonClient(name = "internet-banking-account-opening-service")
public interface ForgotPasswordProxy {

    @PostMapping("/internet-banking-account-opening-service/internet/banking/password/v1/updatePassword")
    ForgotPasswordResponse updatePassword(@Valid @RequestBody ForgotPasswordRequest forgotPasswordRequest);

    @PostMapping("/internet-banking-account-opening-service/internet/banking/password/v1/findByCustomerId")
    FindByCustomerIdResponse getCustomeDataByCutomerId(@Valid @RequestBody FindByCustomerIdRequest findByCustomerIdRequest);

}
```

- ✓ As we are hitting the consumer service via ZuulApiGateWay we need to specify the **@FeignClient** name as ZuulApiGateWay application name
- ✓ As we are hitting the consumer `internet-banking-account-opening-service` we need to specify the **@RibbonClient** name as Consumer application name.
- ✓ In the **@PostMapping** we need to specify the url mapping as consumer name along with its controller path. Below is the consumer application controller,

```
@RestController
@RequestMapping("/internet/banking/password/v1")
@Slf4j
public class UpdatePasswordController {
```

16. What is the use of Netflix Ribbon Client?

It provides the client-side balancing algorithm. It uses a Round Robin Load Balancing. It automatically interacts with Netflix Service Discovery (Eureka) because it is a member of the Netflix family. The important point is that when we use Feign, the Ribbon also applies.

17. Explain the features of Netflix Ribbon Client?

- ✓ Load balancing
- ✓ Fault tolerance
- ✓ Multiple protocols (HTTP, TCP, UDP) support in an asynchronous and reactive model

18. What is meant by load balancing and give different types of load balancing?

Load balancing is defined as the methodical and efficient distribution of network or application traffic across multiple servers in a server farm. Each load balancer sits between client devices and backend servers, receiving and then distributing incoming requests to any available server capable of fulfilling them.

There are two types of load balancing,

Server Side Load Balancing: Server side load balancing is a monolithic it applies between the client and the server. It accepts incoming network, application traffic, and distributes the traffic

across the multiple backend servers by using various methods. The middle component is responsible for distributing the client requests to the server.

Client-Side Load Balancing: The client holds the list of server's IPs so that it can deliver the requests. The client selects an IP from the list, randomly, and forwards the request to the server.

19. **Give an example for Spring Cloud Sleuth?**

Responsible to maintain the unique id from request start to completion.

The Spring Cloud Sleuth token has the following components:

Application name: The name of the application that is defined in the properties file.
Trace Id: The Sleuth adds the Trace Id. It remains the same in all services for a given request.
Span Id: The Sleuth also adds the Span Id. It remains the same in a unit of work but different for different services for a given request.

- ✓ Add below dependency in pom.xml file

```xml
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-sleuth</artifactId>
</dependency>
```

- ✓ Added below method in RestController

```java
@RequestMapping("/callhome") public String callHome() {
    LOG.log(Level.INFO, "calling home");
    return restTemplate.getForObject("http://localhost:8080", String.class);
}
```

- ✓ In the logger we can see this after method execution

```
[slueth-sample,44462edc42f2ae73,44462edc42f2ae73,false] 13978 --- [nio-8080-exec-1] com.example.SleuthSampleApplication : calling home
```

- ✓ In the above logger Boolean value indicates whether the span is exported to Zipkin server or not.

20. **Which design patterns followed by micro services?**

- ✓ API Gateway Pattern
- ✓ Aggregator Pattern

- ✓ Service Discovery Pattern
- ✓ Circuit Breaker Pattern
- ✓ Asynchronous Messaging
- ✓ Chain of Responsibility Pattern
- ✓ Database Pattern

21. Explain about ZuulApiGatewayFilter methods?

ZuulFilter class has four abstract methods that are listed below,

shouldFilter(): The shouldFilter() method checks the request and decides whether filter to be executed or not.
run(): The run() method invokes, if both !isFilterDisabled() and shouldFilter() methods returns true.
filterType(): The filterType() method classify a filter by type.
There are four types of standard filters in Zuul:
pre for pre-routing filtering
route for routing to an origin
post for post-routing filters
error for error handling
Zuul also supports a static type for static responses. Any filter type can be created or added and run by calling the method runFilters(type).
filterOrder(): The filter order must be defined for a filter. Filters may have the same filter order if the precedence is not important for the filters. The filter order does not need to be sequential.

22. Explain about Load Balancing with Zuul?

When Zuul receives a request, it picks up one of the physical locations available and forwards requests to the actual service instance. The whole process of caching the location of the service instances and forwarding the request to the actual location is provided out of the box with no additional configurations needed.

23. Can Zuul Edge Server be used without Eureka / Ribbon?

Yes, it is totally possible by doing the below changes application.yml file of ZuulAPiGateWay application.

```
zuul:
  routes:
    yourService:
      path: /yourService/**
      serviceId: yourService

ribbon:
  eureka:
    enabled: false

yourService:
  ribbon:
    listOfServers: localhost:8080
```

24. What are all the tools you are using for Micro Services In your project?

Kibana: For Logs Monitoring
AppDynamics: For application status monitoring
Eureka Dashboard: For checking the application status and health check

25. In which cases microservice architecture best suited?

Microservice architecture is best suited for desktop, web, mobile devices, Smart TVs, Wearable, etc.

26. How Do You Override a Spring Boot Project's Default Properties?

This can be done by specifying the properties in application.properties.
For example – In Spring MVC applications, you have to specify the suffix and prefix. This can be done by entering the properties mentioned below in application.properties.

For suffix – spring.mvc.view.suffix: .jsp
For prefix – spring.mvc.view.prefix: /WEB-INF/

27. What Do You Mean By End-To-End Testing Of Microservices?

End-to-end testing validates every process in the workflow is functioning correctly. It also ensures that the system works together as a whole and satisfies all requirements.